Adrien Protzel

Query Evaluation and Optimization

1. Cost of Algorithms: Consider the natural join of relations R(A, B) and S(A, C). None of the relations has any indexes. Assume that R and S have 80,000 and 20,000 pages, respectively. The cost of a join is its number of I/Os accesses. For each of the following settings, find the fastest join algorithm to compute the join of R and S and provide its cost. You should choose the join algorithms in your answers from the ones taught in the class. If the algorithms need to sort the relations, they must use the two-pass multi-way merge sort.

Join	Formula	I/O's
In memory nested loops join	T(R) * T(S)	Does not fit in memory
In memory sort-merge join	T(R) logT(R) + T(S) logT(S) + T(R) + T(S)	Does not fit in memory
In-memory hash join	T(R) + T(S)	Does not fit in memory
Page oriented nested loops join	B(R) + B(R) * B(S)	1,600,080,000
Block nested loops join	B(R) + (B(R) / M-2) * B(S)	10,900,000
Index nested loops join	B(R) + T(R) * Index I/O + B(S)	No indexes exist
Sort-merge join (two pass)	5B(R) + 5B(S)	500,000
Optimized sort-merge join (two pass)	3B(R) + 3B(S)	300,000
Hash join	3B(R) + 3B(S)	300,000
Recursive partitioning for hash join	More than 3B(R) + 3B(S)	>300,000

   a. Assume that there are 150 buffers (pages) available in the main memory to execute the join. M^2 = 22,500
      i. Recursive partitioning for hash join is the best feasible option as the other I/O's are too large to fit in memory and the partitioning breaks the relations into smaller partitions for hashing.

   b. Assume that there are 100 buffers (pages) available in the main memory to execute the join. M^2 = 10,000

i. Same as (a), Recursive partitioning for hash join is the best feasible option because of memory constraints

c. Assume that there are 320 buffers (pages) available in the main memory to execute the join. M^2 = 102400
   i. The Optimized sort-merge join (two pass) and Hash join are able to fit in memory and are tied for best cost.

2. Query Evaluation Algorithms
   a. Assume that all tuples of relation R(A, B) fit in the available main memory but relation S(A, C) is too large to fit in the main memory. Find a fast join algorithm, i.e., an algorithm with the lowest number of I/O access, for the natural join of R and S. Justify that your proposed algorithm is the fastest possible join algorithm to compute the natural join of R and S.
      i. Index nested loops join with R in memory is the fastest possible join algorithm in this case, with a cost of B(S) I/Os, because it leverages the fact that R fits in memory and avoids any unnecessary disk operations.

   b. Consider again the relation and setting explained in part (a). Assume that there is a clustered index on attribute A of relation S. Explain whether or how this will change your answer to part (a).
      i. The best algorithm remains the same (hash R in memory, scan S), but the clustered index on S(A) offers an alternative that may be competitive or better if R is very small and index lookups are efficient.

   c. Consider relations R(A, B) and S(A, C) that each have 1 million tuples. Each relation is too large to fit in the main memory. The number of tuples in the result of the natural join of R and S is 100,000. A data scientist wants to compute 1000 (sample) tuples of the natural join of R and S very fast. Since it is too time-consuming to compute the full natural join of R and S, the data scientist selects 1% of tuples in relation R and 1% of tuples in relation S and computes their join. Explain whether this algorithm always returns the desired number of joint tuples. If it does not, propose an efficient algorithm that returns the desired number of joint tuples without computing the full natural join of R and S.
      i. No, this algorithm does not always return 1000 join tuples. Selecting 1% of R and 1% of S gives only a small chance that matching join keys will appear in both samples. Since the join result depends on matching values, most sampled tuples won't join, especially if join keys are spread out.

   d. Let us define two strings similar if they contain the same set of characters. Each shared character between similar strings appears the same number of times in each string. For example, "Corvallis" and "Sillavroc" are similar but "Corvallis" and "Corvalis" are not similar. Consider relations R(A, B) and S(A, C) where

attributes B and C are strings of characters, e.g., varchar data type. Propose a join algorithm that efficiently joins R(A, B) and S(A, C) based on the similarity of attributes B and C, i.e., we like to join pairs of tuples in R and S whose values in attributes B and C are similar.

i. Let us convert each string in tables B and C into a pattern that shows how many times each letter in the string appears. For example, “aabc” will become “a2b1c1”. We will then compare the pattern strings and then join the tables based on the matching strings.

3. Query Optimization: Logical Plans: Consider the following relations:

R(A, B, C, D, E), S(F, D), T(G, B, D, H), U(I, J, K), V(L, J, M), W(L, J, N).

The underlined attributes are the primary keys of their relations. Suggest an optimized logical query plan for the following query and explain why your proposed plan is generally faster than other possible plans. You should find the best guess(es) for the join order in your plan without knowing the statistics of the join attributes and base relations. Your answer may not always be faster than other plans, but it should run faster than other plans for most input relations.

SELECT B, C
FROM R, S, T, U, V, W
WHERE R.D = S.D and R.D = T.D and R.B = T.B and U.J <= V.J and U.J = W.J and R.E <= 200 and W.N <= 100

1. Apply the reduction filters, R.E <= 200 and W.N <= 100
2. Join R.D = S.D
3. Join R.D = T.D and R.B = T.B
4. Join U.J = W.J
5. Join U.J where <= V.J
6. Keep only B and C

4. Query Optimization: Cost Estimation: Consider relations: R(A, B) with 100,000 tuples and S(A, C) with 10,000 tuples. Use Selinger-style, i.e., System-R style, cost estimation formulas to answer the following questions.
    a. We want to select tuples from R that satisfy (A=10) and (B=10). We know that V(R, A) = 100. We do not have any information about the number of distinct values or the range of values in B. What is a reasonable estimate for the number of tuples in the resulting query?
        i. R has 100,000 tuples, A has 100 distinct values. So A=10 is about 1/100 of the tuples.
        ii. 100000 * 1/100 = 1000 tuples in R are A=10
        iii. We guess B has 10 distinct values of B=10 => 1/10

iv. 100000 * 1/100 * 1/10 = 100 tuples in R is A=10 and B=10

b. We want to compute the natural join of R and S. We know that V(R, A) = 100 and V(S, A) = 1000. What is a reasonable estimate for the number of tuples in the resulting query?
   i. Natural join: R join S = (|R| * |S|) / max(V(R, A), V(S, A))
   ii. |R| = 100000, |S| = 10000, V(R,A)=100, V(S,A)=1000
   iii. R join S = 1000000 tuples